(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 961 504 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
01.12.1999 Bulletin 1999/48

(51) Int. Cl.[6]: **H04N 9/80**

(21) Application number: **99108641.4**

(22) Date of filing: **12.05.1999**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(30) Priority: **25.05.1998 DE 19822975**

(71) Applicant:
DEUTSCHE THOMSON-BRANDT GMBH
78048 Villingen-Schwenningen (DE)

(72) Inventors:
- **Winter, Marco**
  30173 Hannover (DE)
- **Schiller, Harald**
  30539 Hannover (DE)

(74) Representative:
Schäferjohann, Volker Willi et al
Deutsche Thomson-Brandt GmbH,
Licensing & Intellectual Property,
Karl-Wiechert-Allee 74
30625 Hannover (DE)

(54) Method and apparatus for the recording and reproduction of video and/or audio signals

(57) A method for the recording and reproduction of video and/or audio signals is proposed in which an additional information item, e.g. a title information item, is additionally recorded, which information item serves for subsequent insertion into a video picture during reproduction. According to the invention, the additional information data are converted, thereby producing an executable sub-picture unit (SPU), which is recorded in a corresponding sub-picture data pack (SP_PCK) in addition to the data packs for the video/audio signal (V_PCK, A_PCK). Thus, it is possible to record the additional information data separately without difficulty, without necessitating a change to the recording format to the effect that special data packs for the additional information items are included in the planning.

The invention also relates to a corresponding recording and reproduction apparatus.

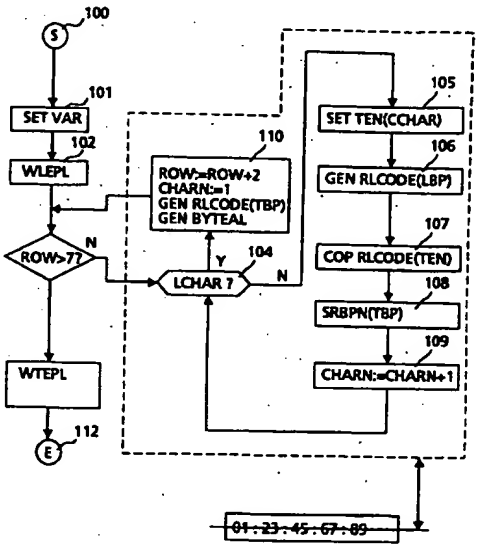

Fig.7

EP 0 961 504 A2

to have inserted, since even the existing DVD players already provide the option of selecting subtitles in different languages.

[0007] It is likewise advantageous that the outlay on circuitry at least for the reproduction of the additional information items is very low, since for this purpose use is made of the same sub-picture decoding unit which is also used otherwise during the decoding of the sub-picture unit for subtitles. An additional circuit can therefore be omitted in this case.

[0008] Advantageous developments and improvements of the method specified in Claim 1 are possible by virtue of the measures evinced in the further Claims 2 - 10. Claim 2 characterizes the structure of the sub-picture unit more precisely. In particular, the fact that the sub-picture unit comprises a compressed bit map for the additional information items is defined in this claim. This property is advantageous because a considerable amount of memory space can be saved as a result. The memory space obtained in this way can be used for concomitantly storing a plurality of different additional information items which can be inserted alternatively. Furthermore, the fact that a sub-picture unit comprises a table with control commands for the insertion representing the additional information item is defined in Claim 2. Commands of this type are already known from the DVD Standard. They allow diverse possibilities for configuring the insertions. Thus, graphical symbols can also be inserted, and it is possible to use different types of highlighting for characters etc. Therefore, the additional information item is not just restricted purely to text information, it can also relate to specific symbols and graphics.

[0009] By way of example, in accordance with Claim 3, run length coding can advantageously be used for the compression of the bit map. This is simple to implement.

[0010] According to Claim 4, the additional information item can be input by an operator or else be machine-generated, such as e.g. in the case of insertion of the playing time or of the time of day and of the date.

[0011] A very advantageous measure is specified in Claim 6, which prescribes the provision of a table with characters that are already run length-coded, from which table the characters required for the additional information items are selected and are combined to form the resultant bit map of the sub-picture unit. This considerably simplifies the routine coding process. The concrete specifications which are made in the table for each character are specified in Claim 7. A number of different additional information items which can be recorded are specified in Claim 9.

[0012] A programming option which enables the user to prescribe the order in which the additional information items that he desires are to be successively inserted can also advantageously be provided in the method according to the invention.

[0013] A corresponding recording and reproduction apparatus for the recording and reproduction of video and/or audio signals in which an additional information item can additionally be recorded is specified in Claim 11.

Drawings

[0014] Exemplary embodiments of the invention are illustrated in the drawings and are explained in more detail in the description below. In the drawings:

Fig. 1	shows the division of the recorded data stream of a DVD into video data packs, audio data packs and sub-picture data packs;
Fig. 2	shows the structure of a sub-picture data pack;
Fig. 3	shows the structure of a sub-picture unit;
Fig. 4	shows the division of a sub-picture unit into a number of sub-picture data packs;
Fig. 5	shows a block diagram concerning the recording part of the recording and reproduction apparatus;
Fig. 6	shows a block diagram of the reproduction part of the recording and reproduction apparatus;
Fig. 7	shows a flow diagram for a program for generating the compressed bit map of a sub-picture unit;
Fig. 8	shows a flow diagram for the conversion of the additional information item into a sub-picture unit;
Fig. 9	shows a rough flow diagram for a program for generating the sub-picture data packs from the additional information items;
Fig. 10	shows a program listing for an example of a program for real-time sub-picture data pack generation;
Fig. 11	shows three examples of function calls relating to the program in Fig. 10;
Fig. 12	shows a fourth, more detailed example of a function call, and
Figs. 13a-13i	show eight examples of applications for additional information insertions.

Exemplary embodiments of the invention

[0015] Figure 1 illustrates the structure of a video object set (VOBS) in accordance with the DVD Standard mentioned in the introduction. A video object set is part of the logical data structure of a DVD video disc which is described in very precise detail in the DVD Standard. Further data units are also stored on a DVD disc, but they will not be discussed in more detail since they do not comprise the video and audio data and the data for a sub-picture unit which are important for the invention

correct physical order and can be used directly for recording on the DVD disc 51. The reference numeral 41 designates the data buffer for the audio data. The data located therein are processed by the audio coding circuit 44, which may be, for example, either an MPEG audio encoding circuit or a DOLBY AC3 audio encoding circuit. The data thus generated are made available in turn to the data formatting unit 49.

[0020] The reference numeral 47 designates a keyboard unit. The latter is connected to a microcontroller 46. The user can carry out inputs via the keyboard unit 47. In particular, he can input e.g. a desired title which is to be concomitantly recorded. Of course, any other desired inputs are equally well possible by this means. In the microcontroller 46, the data that have been input are likewise ordered logically again and then forwarded to a buffer store 42. The data located therein are revised by a sub-picture coding unit 45. A sub-picture unit for the data that have been input is generated in the sub-picture coding unit 45. The said sub-picture unit is then forwarded to the data formatting unit 49. The formation of the data packs for the video data packs, audio data packs and sub-picture data packs can preferably be performed in the data formatting unit 49.

[0021] A real-time clock 48 is additionally connected to the microcontroller 46. Time-of-day data and also date details can be communicated by the said real-time clock. These data and details are then also converted by the microcontroller 46 and can be used for special insertions (sub-picture insertions). It has not been mentioned heretofore that the recorded data are protected by multiple error protection. This task can also be undertaken by the formatting unit 50.

[0022] As an alternative, the sub-picture encoding unit 45 may also be integrated in the microcontroller 46 if the latter is correspondingly powerful enough.

[0023] The block diagram of Figure 6 is explained below; Figure 6 illustrates the essential components used during reproduction of the recorded data. The reference numeral 58 designates a serial data input, where a bit stream is present which contains both video data and audio data and the data for the sub-pictures. The data are supplied by an optical storage disc DVD 51. The incoming data are then initially subjected to error detection and correction in a correction unit 60. The data subsequently pass into a separator circuit 61, in which the video, audio and sub-picture data, which are still mixed together, are separated and respectively transferred accordingly either to a video decoding unit 62, a sub-picture decoding unit 63 or to an audio decoding unit 68. The decoded video and sub-picture data are made available to a multiplexing unit 64. The multiplexing unit 64 is controlled by the sub-picture decoding unit 63. At the output of the multiplexing unit 64, the data for the individual pixels of the video picture are successively input into a TV signal decoding device 65. The standard-conforming luminance and chrominance signals (Y, C) are output in digital form at the outputs of the TV signal encoding device (PAL, SECAM, NTSC). These signals are subsequently converted into analog signals in the D/A conversion unit 66 and forwarded to corresponding outputs 70, 71. The associated audio signal is already generated in a standard-conforming manner in the decoding device 68 and converted into an analog audio signal (only a mono signal is illustrated in this case) in the D/A conversion unit 69. This audio signal is made available at the output 73.

[0024] In another embodiment, the audio signal may also be output in digital form. This signal must be processed further in an external decoder in that case.

[0025] On the other hand, the embodiment may also be such that the analog luminance and chrominance signals and the audio signal are modulated onto different carriers in a modulation unit 67 and output as a corresponding V signal via just one output 72.

[0026] The reference numeral 59 designates an additional input for the keyboard unit 47. The input can also be embodied as an infrared input if the keyboard is integrated on a remote control. The microcontroller 46 serves to control the units 60, 61, 62, 63, 68. After a corresponding selection command has been input, the microcontroller 46 sets e.g. the sub-picture decoding unit 63 in such a way that it decodes only that sub-picture unit in accordance with the input. All other sub-picture units that may be present are then ignored. The sub-picture decoding unit 63 then inputs the decoded data into the bit stream for the entire video picture at the preprogrammed locations. For this purpose, the sub-picture decoding unit 63 drives the multiplexing unit 64 with correct timing in accordance with the horizontal and vertical sync pulses, input into the said unit by the TV signal encoding device 65, and the pixel clock signal. Further details concerning the structure and the method of operation of such a sub-picture decoding unit 63 are contained in EP-A-0 725 541. In this regard, therefore, reference is expressly made to this document as well.

[0027] Since the essential difficulty of the invention resides in generating the sub-picture unit under real-time conditions for a title insertion, an example of a program for sub-picture data pack generation is explained below with reference to the flow diagrams of Figures 7 - 9. The compressed bit map which is used for the sub-picture unit is generated from a prescribed ASCII text by means of the program according to Figure 7.

[0028] A completely executable sub-picture unit is generated from the compressed bit map generated previously by means of the program in accordance with Figure 8. Then, finally, a complete sub-picture data pack is generated with the aid of the sub-picture unit generated by means of the program in accordance with Figure 9. This sub-picture data pack can then be inserted directly into the sector stream of a video object unit VOBU. The precise way in which this is to be done is already disclosed by the DVD Standard and, therefore, need not be explained in any further detail here.

[0029] First of all the function for generating the com-

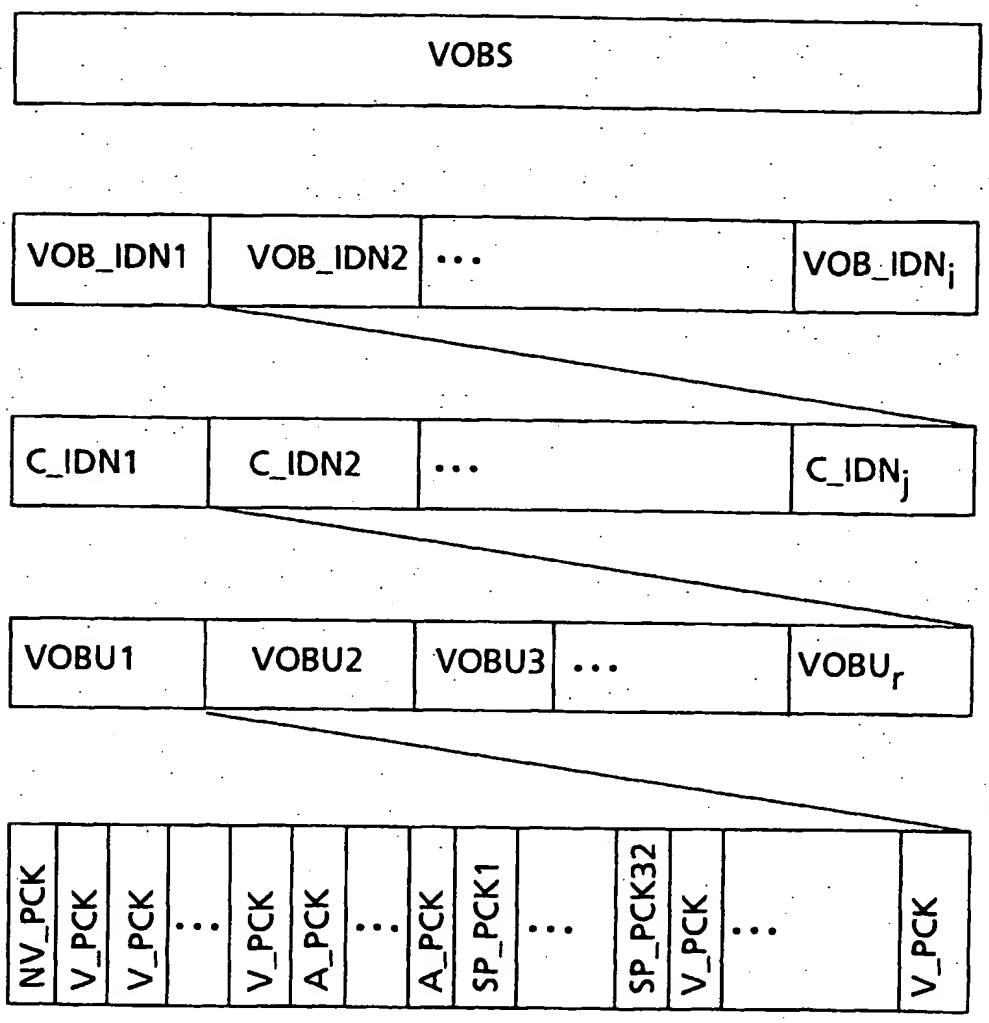
VOBS
VOB_IDN1
VOB_IDN2
• • •
VOB_IDN$_i$
C_IDN1
C_IDN2
• • •
C_IDN$_j$
VOBU1
VOBU2
VOBU3
• • •
VOBU$_r$
NV_PCK
V_PCK
V_PCK
• • •
V_PCK
A_PCK
• • •
A_PCK
SP_PCK1
• • •
SP_PCK32
V_PCK
• • •
V_PCK

Fig.1

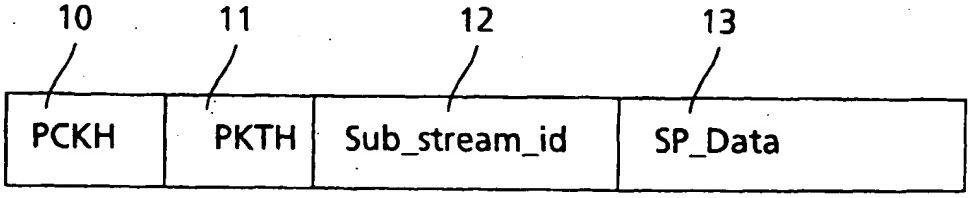
10
11
12
13
PCKH
PKTH
Sub_stream_id
SP_Data

Fig.2

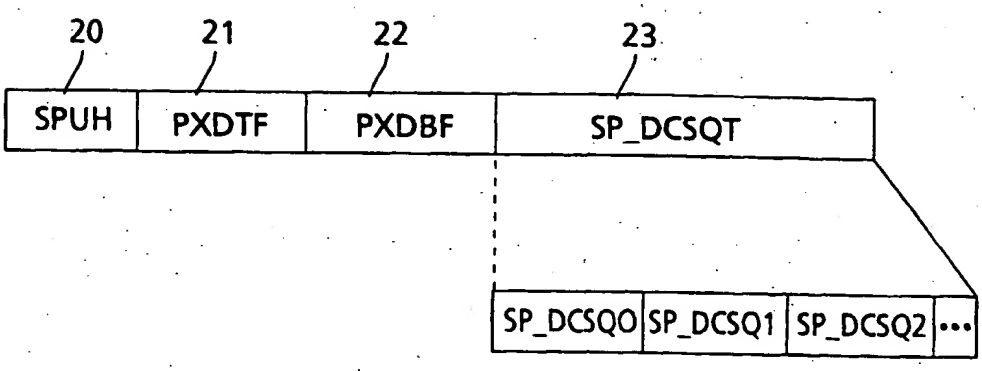
20
21
22
23
SPUH
PXDTF
PXDBF
SP_DCSQT
SP_DCSQ0
SP_DCSQ1
SP_DCSQ2
...

Fig.3

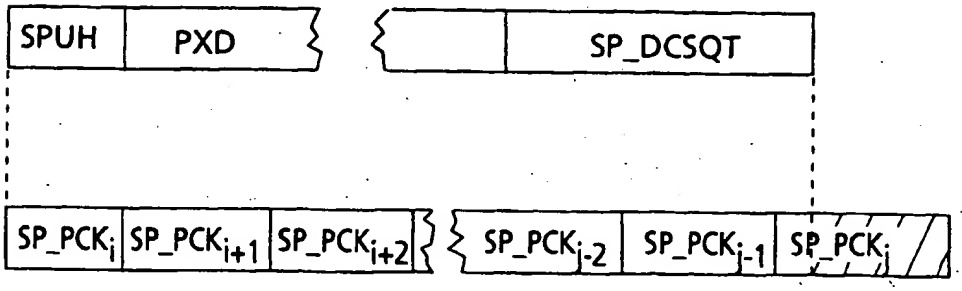
SPUH
PXD
SP_DCSQT
SP_PCK$_i$
SP_PCK$_{i+1}$
SP_PCK$_{i+2}$
SP_PCK$_{j-2}$
SP_PCK$_{j-1}$
SP_PCK$_j$

Fig.4

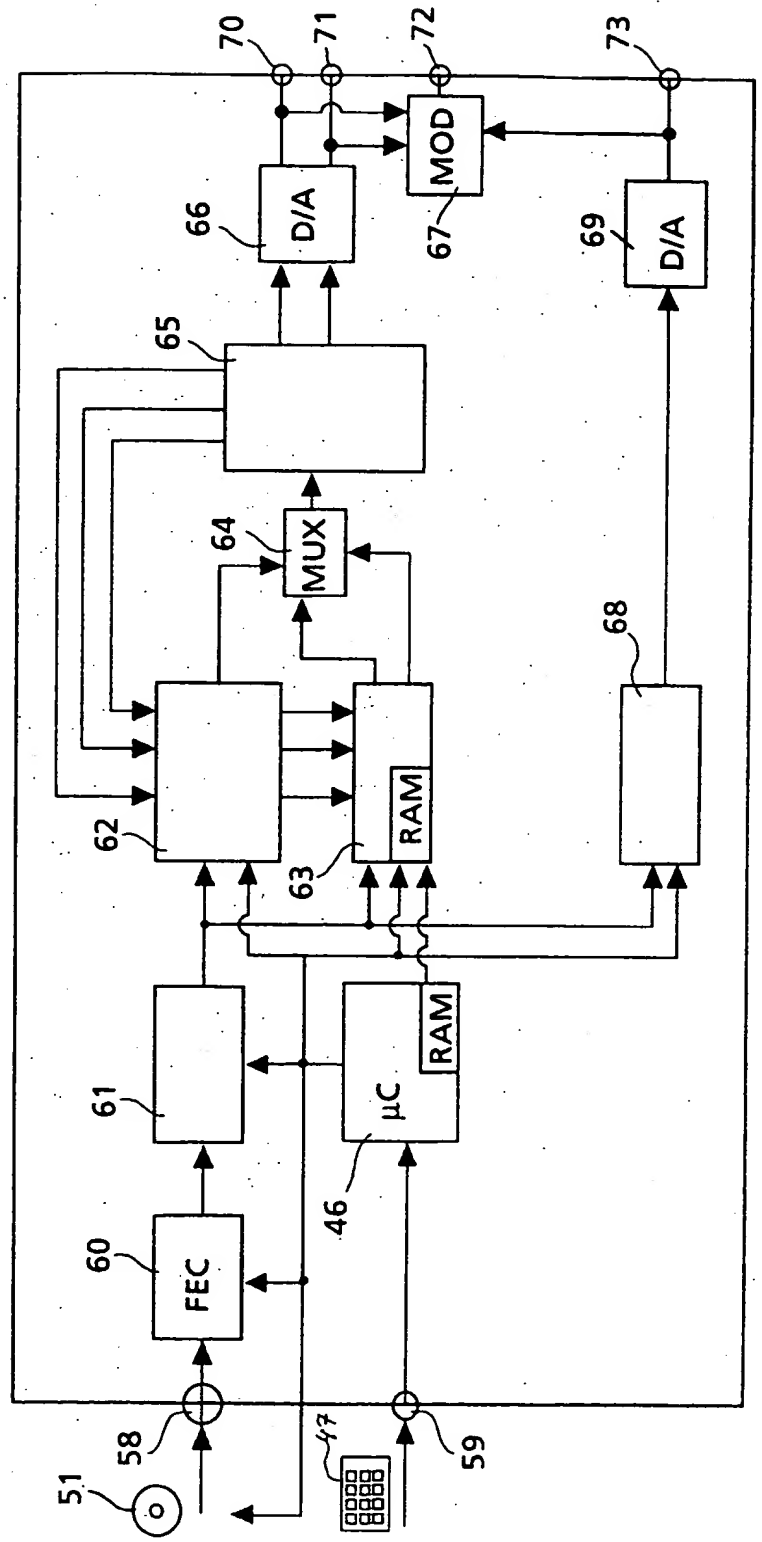
70
71
72
73
MOD
D/A
D/A
66
67
69
65
MUX
64
68
RAM
62
63
RAM
μC
61
46
FEC
60
58
59
51

Fig.6

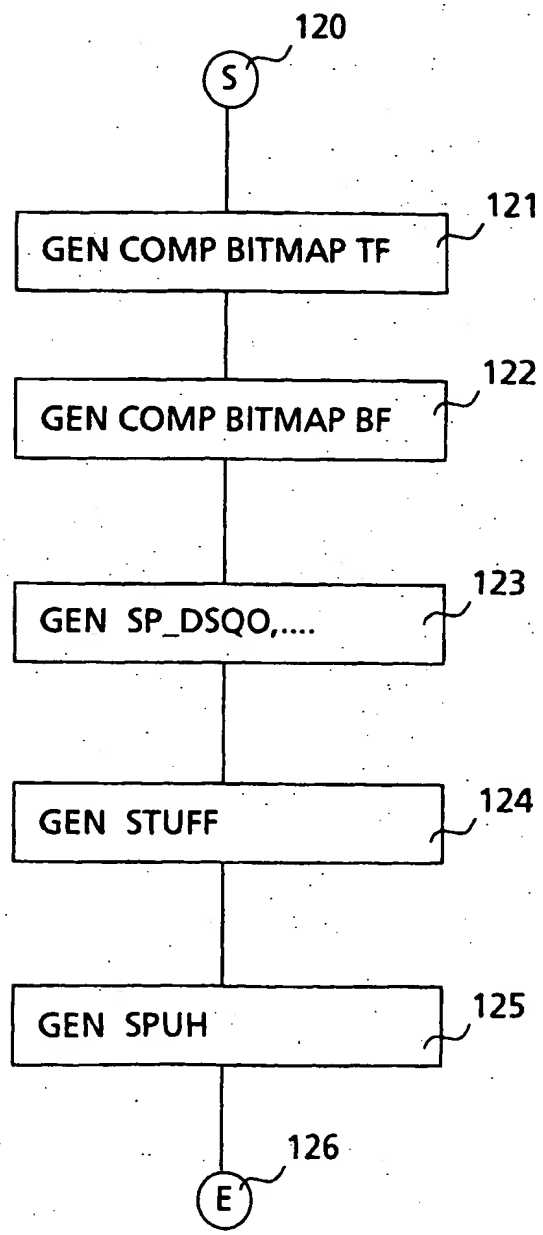
120
S
121
GEN COMP BITMAP TF
122
GEN COMP BITMAP BF
123
GEN SP_DSQO,....
124
GEN STUFF
125
GEN SPUH
126
E

Fig.8

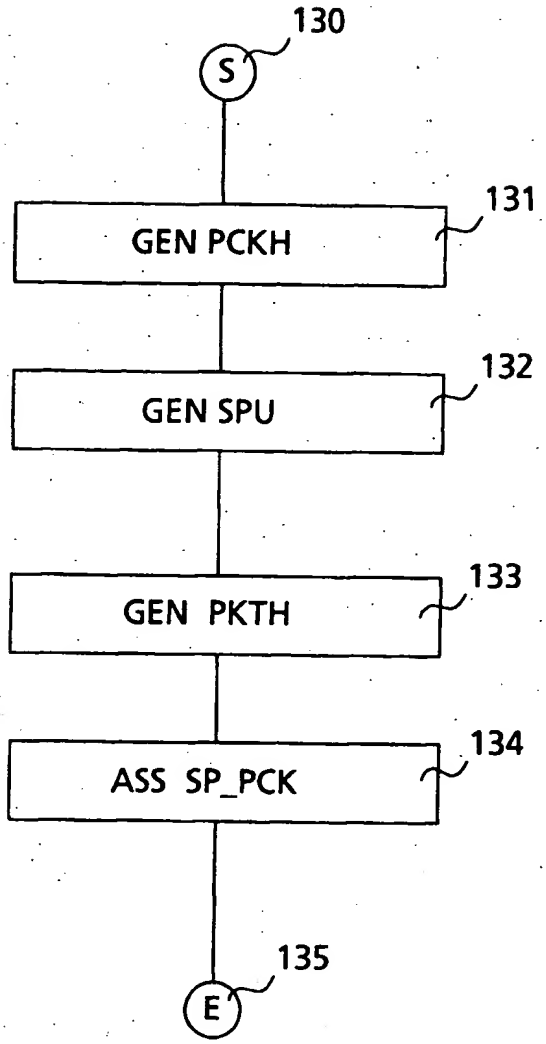

Fig.9

```
    {1, 1, 1,   {0x90,    0   } }, /*   .11.  */
    {3, 0, 1,   {0x54, 0x50   } }, /*   1.1.  */
    {1, 2, 1,   {0x50,    0   } }, /*   .1.   */
    {1, 2, 1,   {0x50,    0   } }, /*   ..1.  */
    {1, 2, 1,   {0x50,    0   } }, /*   ..1.  */
    {1, 1, 0,   {0xD0,    0   } } /*   .111  */
},  {/*** 2  ***/
    {1, 1, 1,   {0x90,    0   } }, /*   .22.  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   2..2  */
    {1, 3, 0,   {0x50,    0   } }, /*   ...2  */
    {1, 2, 1,   {0x50,    0   } }, /*   ..2.  */
    {1, 1, 2,   {0x50,    0   } }, /*   .2..  */
    {1, 0, 3,   {0x50,    0   } }, /*   2...  */
    {2, 0, 0,   {0x11,    0   } } /*   2222  */
},  {/*** 3 ***/
    {1, 1, 1,   {0x90,    0   } }, /*   .33.  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   3..3  */
    {1, 3, 0,   {0x50,    0   } }, /*   ...3  */
    {1, 2, 1,   {0x50,    0   } }, /*   ..3.  */
    {1, 3, 0,   {0x50,    0   } }, /*   ...3  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   3..3  */
    {1, 1, 1,   {0x90,    0   } } /*   .33.  */
},  {/*** 4 ***/
    {1, 0, 3,   {0x50,    0   } }, /*   4...  */
    {1, 0, 3,   {0x50,    0   } }, /*   4...  */
    {3, 0, 1,   {0x54, 0x50   } }, /*   4.4.  */
    {3, 0, 1,   {0x54, 0x50   } }, /*   4.4.  */
    {2, 0, 0,   {0x11,    0   } }, /*   4444  */
    {1, 2, 1,   {0x50,    0   } }, /*   ..4.  */
    {1, 2, 1,   {0x50,    0   } } /*   ..4.  */
},  {/*** 5 ***/
    {2, 0, 0,   {0x11,    0   } }, /*   5555  */
    {1, 0, 3,   {0x50,    0   } }, /*   5...  */
    {1, 0, 3,   {0x50,    0   } }, /*   5...  */
    {1, 0, 1,   {0xD0,    0   } }, /*   555.  */
    {1, 3, 0,   {0x50,    0   } }, /*   ...5  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   5..5  */
    {1, 1, 1,   {0x90,    0   } } /*   .55.  */
},  {/*** 6 ***/
    {1, 1, 1,   {0x90,    0   } }, /*   .66.  */
    {1, 0, 3,   {0x50,    0   } }, /*   6...  */
    {1, 0, 3,   {0x50,    0   } }, /*   6...  */
    {1, 0, 1,   {0xD0,    0   } }, /*   666.  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   6..6  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   6..6  */
    {1, 1, 1,   {0x90,    0   } } /*   .66.  */
},  {/*** 7 ***/
    {2, 0, 0,   {0x11,    0   } }, /*   7777  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   7..7  */
    {1, 3, 0,   {0x50,    0   } }, /*   ...7  */
    {1, 2, 1,   {0x50,    0   } }, /*   ..7.  */
    {1, 1, 2,   {0x50,    0   } }, /*   .7..  */
    {1, 1, 2,   {0x50,    0   } }, /*   .7..  */
    {1, 1, 2,   {0x50,    0   } } /*   .7..  */
},  {/*** 8 ***/
    {1, 1, 1,   {0x90,    0   } }, /*   .88.  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   8..8  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   8..8  */
    {1, 1, 1,   {0x90,    0   } }, /*   .88.  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   8..8  */
    {3, 0, 0,   {0x58, 0x50   } }, /*   8..8  */
```

A

```
  { 1, 1, 1,  {0x90,        0    }}  /*    .88.    */
},{ /*** 9 ***/
  { 1, 1, 1,  {0x90,        0    }},  /*    .99.    */
  { 3, 0, 0,  {0x58,     0x50    }},  /*    9..9    */
  { 3, 0, 0,  {0x58,     0x50    }},  /*    9..9    */
  { 1, 1, 0,  {0xD0,        0    }},  /*    .99     */
  { 1, 3, 0,  {0x50,        0    }},  /*    ...9    */
  { 1, 3, 0,  {0x50,        0    }},  /*    ...9    */
  { 1, 1, 1,  {0x90,        0    }}   /*    .99.    */
},{ /*** : ***/
  { 0, 4, 0,  {    0,       0    }},  /*    ....    */
  { 0, 4, 0,  {    0,       0    }},  /*    ....    */
  { 1, 1, 2,  { 0x50,       0    }},  /*    *..     */
  { 0, 4, 0,  {    0,       0    }},  /*    ....    */
  { 1, 1, 2,  { 0x50,       0    }},  /*    *..     */
  { 0, 4, 0,  {    0,       0    }},  /*    ....    */
  { 0, 4, 0,  {    0,       0    }}   /*    ....    */
},{ /*** SPACE ***/
  { 0, 4, 0,  {    0,       0    }},
  { 0, 4, 0,  {    0,       0    }},
  { 0, 4, 0,  {    0,       0    }},
  { 0, 4, 0,  {    0,       0    }},
  { 0, 4, 0,  {    0,       0    }},
  { 0, 4, 0,  {    0,       0    }},
  { 0, 4, 0,  {    0,       0    }}
 }
};

/* generates byte alignment for the compressed bitmap of a SP line:  */
#define putAlign( )  if( !tmppos); else putHBit( (Uint32) 0, 8-tmppos)

/* puts the <n> most significant bits of the 32 bit value <b>
 * into buffer <bitmap>:  */
static void putBit( Uint32 b,  Int32 n) {

 */ significant  bits in <b> are on the left side of Uint32 word */

 while( --n>=0 )  {
  tmppos++;
  tmpbuf «= 1;
  if( b&(unsigned long)0x80000000 )  tmpbuf++;
  b «= 1;
  if(  tmppos == 8 )  {
    *bitmap++ = tmpbuf;
    tmppos = 0;
    tmplen++;
   }
 }

} /* putBit( ) */

/* puts the <n> most significant bits of the first 8 bits of
 * value <b> into buffer <bitmap>:  */
static void putHBit( Uint32 b,  Int32 n )  {
 putBit( b«24,          /* significant bits to left long word limit */
       n );
} /* putHBit( )  */
/* puts the <n> least significant bits of the 32 bit value <b>
 * into buffer <bitmap>:  */
static void putLBit( Uint32 b,  Int32 n )  {
 putBit( b«(32-n),      /* significant bits to left long word limit */


       n );
 } /* putLBit( )  */
```

```
else                                        /* it's a special char.   */
 switch( *text ) {
 case ':':  character = 10; break;
 default:  character =  11; break;          /* use space as default */

/* get leading background pixel: */
n = lastNBackgroundPel + characterSet[ character ] [ row ] .nBeginningPel;

/* when there are pattern pixel for this row then encode those;   */
if( characterSet[ character ] [ row ] .nNibbel ) {

  /* encode leading background pixel (if any):   */
  while ( n ) {   /* generate run-length code    */
    if ( n<16)
    if ( n<4)
      putLBit( n, 2 );   /* put the 2 LSB of n onto bitmap */
    else
      putLBit( n, 6 );
   else
      if( n<64,)
      putLBit( n, 10 );
    else if( n<256)
      putLBit( n, 14 );
    else
      putLBit( 255, 14);
   putHBit( 0, 2 );
   if( n >= 256 ) n -= 255;
   else    n  = 0;
  }

  /* insert preencoded character run-length code:    */
  for( n=0; n<characterSet[ character ] [ row ] .nNibbel; n++ ) {
   if( n&l1 )    /* odd */
     putLBit( characterSet[ character ][ row ].runLengthcode[n»1],    4 );
   else /* even */
     putHBit( characterSet[ character ][ row ].runLengthcode[n»1],    4 );
  }
  /* prepare encoding of trailing background pixel    (if any): */
  lastNBackgroundPel = characterSet[ character ] [ row ] .nRemainingPel;

  } else { /* there are no pattern pixel to encode */

  /* prepare encoding of background pixel (if any):   */

  lastNBackgroundPel = n + characterSet[ character ] [ row ] .nRemainingPel
 }

 /* insert space between 2 characters: */
 lastNBackgroundPel += SPACE_BETWEEN_2_CHARACTERS;
 text++; /* next character */
} /* while */

/* encode the remaining background pels of the current SP row:    */
n = lastNBackgroundPel + offsetRight - SPACE_BETWEEN_2_CHARACTERS
if( n ) {
 if( n<16 )
   if( n<4)
     putLBit( n, 2 );  /* put the 2 LSB of n onto bitmap */
   else
```

B

```
      putLBit( n, 6 );
    else
     if( n<64 )
      putLBit( n, 10 );
     else if( n<256)
      putLBit( n, 14 );
     else
      putLBit( 0, 14);
    putHBit( 0, 2 );


  /* add nibbel stuffing in order to get the SP line byte aligned: */
  putAlign( );
 } /* for row */

 /* lower background pixel offset of the SP: */
 for( row=(offsetUp+N_ROWS+topField)&1; row<offsetDown; row += 2 ) {
  *bitmap++ = 0;
  *bitmap++ = 0;
 }

 /* return the byte length of the just encoded compressed data */
 return (Uint32) (bitmap - startOfBitmap);

} /* makeCompressedBitmap( ) */

/ ************************************************************************
 *
 *    Extern Function: makeSPU( )
 *
 *    Encodes characters from a text string to complete SPU
 *    Input: pointer to an already allocated buffer for the complete SPU
 *          and a text string which shall be encoded as SP
 *    Output: the encoded SPU
 *
 *    Return value: length of the whole SPU
 *
 ************************************************************************/
extern Uint32 makeSPU ( )
        Uint8   ntsc,           /* 0: 625/50; 1: 525/60 */
        Uint8  *spuBuffer,      /* OUT: already allocated buf fer for the SPU */
        char   *text,           /* this text shall be encoded as SP */
        UInt32  durationInFrames,  /* after this no. of frames
                                    * the SP shall disappear */
        Uint32 starty           /* vertical start position of the SP */
        ) {
 Uint32
  spWide,      /* horizontal size of SP (in pel) */
  spHeight,    /* vertical size of SP (in pel) */
  lenTop,      /* byte length of the top   field compressed bitmap */
  lenBottom,   /* byte length of the bottom field compressed bitmap */
  startx;      /* start column of the SP on screen */
 Uint32
  offset;      /* byte offset relative to the start of the SPU */

 starty &= -1;  /* starty MUST be even */

 /* generate run-length code for top field: */
 lenTop   = makeCompressedBitmap( spUBuffer+4
                                  0,
                                  text,
```

```
                                        8, 8, 3,  3,
                                        &spWide,  &spHeight ) ;

/* generate run-length code for bottom field:   */
lenBottom = makeCompressedBitmap ( spuBuffer+4+lenTop,
                                        1,
                                        text,
                                        8, 8, 3,  3,
                                        &spWide,  &spHeight ) ;

/* place SP at horizontal screen center position:   */
startx =  (720 - spWide) / 2;

/* set offset to start of SP_DCSQT:   */
offset = 4 + lenTop + lenBottom;

/* size of SP_DCSQT <= half of the size of SPU */
while (offset < 30 )  spuBuffer [ offset++ ] = 0;

spuBuffer[0]  =   ((offset+31)&-1)  »   8;       /* size of whole SPU */
spuBuffer[1]  =   ((offset+31)&-1)  &   0xFF;

spuBuffer[2]  = offset »    8;                   /* start of SP_DSQ #0 */
spuBuffer[3]  = offset   &  0xFF;

/*** DCSQ#0:  ***/

spuBuffer[offset    ]  = 0x00;                   /* STM = 0 */
spuBuffer[offset+1]  = 0x00;
spuBuffer[offset+2]  = (offset+24) »      8;     /* ptr to next SP_DCSQ */
spuBuffer[offset+3]  = (offset+24)   & 0xFF;

spuBuffer[offset+4]  = 0x03;    /* SET_COLOR b=0 p=1 e1=2 e2=3 */
spuBuffer[offset+5]  = 0x32;
spuBuffer[offset+6]  = 0x10;

spuBuffer[offset+7]  = 0x04;    /* SET_CONTR (0..15) b=15 p=15 e1=15 e2=15 */
spuBuffer[offset+8]  = 0xFF;
spuBuffer[offset+9]  = 0xFF;

spuBuffer[offset+10]  = 0x05;  /* SET_DAREA */
spuBuffer[offset+11]  = startx»4;
spuBuffer[offset+12]  = ((startx«4) +( (startx+spWide-1)»8)) & 0xF3;
spuBuffer[offset+13]  = (startx+spWide-1) & 0xFF;
spuBuffer[offset+14]  = starty»4;
spuBuffer[offset+15]  =  ((starty«4)+((starty+spHeight-1)»8))   & 0xE3;
spuBuffer[offset+16]  =  (starty+spHeight-1) & 0xFF;

spuBuffer[offset+17]  = 0x06;  /* SETDSPXA */
spuBuffer[offset+18]  = 0x00;
spuBuffer[offset+19]  = 0x04;
spuBuffer[offset+20]  =  (4+lenTop) »   8;
spuBuffer[offset+21]  =  (4+lenTop)  & 0xFF;

spuBuffer[offset+22]  = 0x01;  /* STA_DSP   */

spuBuffer[offset+23]  = 0xFF;  /* CMD_END */

/*** SP_DCSQ#1:        ***/

if( ntsc )  {  /* STM for NTSC:  */
```

```
    spuBuffer[offset+24]  =  (durationInFrames * 3003) » 18;
    spuBuffer[offset+25]  = ((durationInFrames * 3003) » 10) & 0xFF;
  } else {     /* STM for PAL: */
    spuBuffer[offset+24]  =  (durationInFrames * 225) » 14;
    spuBuffer[offset+25]  = ((durationInFrames * 225) »  6) &  0xFF;

  spuBuffer[offset+26]  = spuBuffer[offset+2] ;    /* ptr to Sp_DCSQ#1 */
  spuBuffer[offset+27]  = spuBuffer[offset+3] ;

  spuBuffer[offset+28]  = 0x02; /* STP_DSP   */

  spuBuffer[offset+29]  = 0xFF;  /* CMD_END */

  if( offset&1 )  spuBuffer[offset+30] = 0xFF;  /* stuffing if necessary */

  return (offset+31)  & -1;     /* length of the whole SPU */

} /*   makeSpu( ) */
/**** ************************************************************************
  *
  *    Extern Function: makeSPPack( )
  *
  *    Generates a complete sub-picture pack.
  *    Input: pointer to an already allocated buffer for the complete
  *           2048 main data
  *    Output:  the encoded SP pack
  *
  *    No return value
  *
  *    Note:  this simple version doesn't perform scrambled encoding
  *
  ************************************************************************/

extern void makeSPPack(
       Uint8   ntsc,              /* 0: 625/50;  1: 525/60 */
       Uint8   *packBuffer,       /* OUT: already allocated buffer for the SPU */
       char    *text,             /* this text shall be encoded as SP */
       uint32  durationInFrames,  /* after this no. of frames
                                   * the SP shall disappear */
       Uint32  starty,            /* vertical start position of the SP */
       Uint8   *scr,              /* 6 bytes SCR */
       Uint8   *pts,              /* 5 bytes PTS,must be on top field boundary!  */
       Uint8   firstPacket,       /* 1:first packet of a VOB;  else 0 */
       Uint8   spStreamNumber     /* SP stream number: 0..31 */
       ) {
  Uint32 spuSize;
  Uint32 pesPrivateDataSize=(firstPacket?3:0) ;

  /* insert pack header:  */
  packBuffer[0]  = 0x00;
  packBuffer[1]  = 0x00;
  packBuffer[2]  = 0x01;
  packBuffer[3]  = 0xBA;

  packBuffer[4]  = scr[0];
  packBuffer[5]  = scr[1];
  packBuffer[6]  = scr[2];
  packBuffer[7]  = scr[3];
  packBuffer[8]  = scr[4];
  packBuffer[9]  = scr[5];
```

```
packBuffer[10]   = 0x01;
packBuffer[11]   = 0x89;
packBuffer[12]   = 0xC3;

packBuffer[13]   = 0xF8;
/*  insert SPU:     */
spuSize = makeSpU( ntsc,
                   packBuffer+14+14+1+pesPrivateDataSize,
                   text,
                   durationInFrames,
                   starty );

/* insert packet header:   */
packBuffer[14]   = 0x00;
packBuffer[15]   = 0x00;
packBuffer[16]   = 0x01;
packBuffer[17]   = 0x8D;

packBuffer[18]   = (spuSize+8+1+pesPrivateDataSize) »    8;
packBuffer[19]   = (spuSize+8+1+pesPrivateDataSize)   & 0xFF;

packBuffer[20]   = 0x81;
packBuffer[21]   = 0x80 + (firstpacket != 0);
packBuffer[22]   = 5+pesPrivateDataSize;

packBuffer[23]   = pts[0];
packBuffer[24]   = pts[1];
packBuffer[25]   = pts[2];
packBuffer[26]   = pts[3];
packBuffer[27]   = pts[4];

if( pesPrivateDataSize )  {
  packBuffer[28]   = 0x1E;
  packBuffer[29]   = 0x60;
  packBuffer[30]   = 58;


/* private data sub stream id;   */
packBuffer [28+pesPrivateDataSize]  = 0x20 + (spStreamNumber&0x1F);

/* clear unused sector rest:   */
while( spuSize + 14 + 14 + 1 + pesPrivateDataSize < 2048 )
  packBuffer[ spuSize++ + 14 + 14 + 1 + pesPrivateDataSize ]    = 0;
} /* makeSppack( )  */
```

D

Fig.10

```
/* ------------------------------------------------------------------------------------------
| Copyright (c) 1998, This text is the property of Thomson
| multimedia and shall not be reproduced, copied, or distributed
| without written permission.
------------------------------------------------------------------------------------------ */
```

Note: values like 0xAB indicates hexadecimal values (C syntax).

```
/************************** THE FUNCTION CALL:*********************************/

/* NTSC
 * text:  "01:23",
 * SP shall be active for 123 frames
 * vertical start position of the SP:  400,
 * scr =  { 0x44, 0x00,  0x45,  0x46, 0xD4,  0xAB }
 * pts =  { 0x21, 0x00, 0x11,  0xDD, 0x09 }
 * it's the first SPU in this VOB,
 * it's SP stream #3 :*/
makeSPPack(
  1,                    /* 0: 625/50;  1: 525/60 */
  mainData,             /* ptr to sector main data */
  "01:23",              /* this text shall be encoded as SP */
  123,                  /* after this no.  of frames
                         * the SP shall disappear */
  400,                  /* vertical start position of the SP */
  scr,                  /* 6 bytes SCR */
  pts,                  /* 5 bytes PTS, must be on a top field!  */
  1,                    /* 1:first packet of a VOB,  else 0  */
  3);                   /* SP stream number:  0..31  */

/************************** THE RESULTING PACK:  *********************************/

hex-
addr   :     content of the pack (hex values)
------------------------------------------------------------------------------
0000000: 00 00 01 BA  44 00 45  46 D4 AB 01  89 C3 F8  00  00
0000010: 01 BD 00 76  81 81 08  21 00 11 DD  09 1E 60  3A  23
0000020: 00 6A 00 4C  00 00 00  00 20 58 5C  92 45 85  85  85
0000030: 20 20 58 51  05 2C 51  45 24 20 58  51 05 24  51  45
0000040: 85 20 00 00  00 00 00  00 24 91 45  28 91 09  24  20
0000050: 58 58 54 51  05 1C 51  45 20 20 58  51 05 10  51  45
0000060: 1C 52 00 24  91 0D 20  11 C9 24 00  00 00 00  00  64
0000070: 03 32 10 04  FF FF 05  15 21 7D 19  01 9C 06  00  04
0000080: 00 26 01 FF  01 68 00  64 02 FF 00  00 00 00  00  00
0000090: 00 00 00 00  00 00 00  00 00 00 00  00 00 00  00  00
00000A0 00 00 00 00  00 00 00  00 00 00 00  00 00 00  00  00
0000080: 00 00 00 00  00 00 00  00 00 00 00  00 00 00  00  00

... (the remaining bytes till byte #2048 of the pack are all Zero)
```

```
/ ****************  ONLY THE SPU IN THE PACK:  ***************************/

hex
addr:  content of the pack  (hex values)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

0020: ######################## SPUH:
0020: 00 6A                    #  SPDSZ     = 106 = 0x6A
0022: 00 4C                    #  SP_DCSQTA = 76 = 0x4C

0024: ######################## PXD:
0024: 00 00 00 00  20  58 5C 92 45  85  85  85 20  20  58  51
0034: 05 2C 51 45  24  20 58 51 05  24  51  45 85  20  00  00
0044: 00 00 00 00  24  91 45 28 91  09  24  20 58  58  54  51
0054: 05 1C 51 45  20  20 58 51 05  10  51  45 1C  52  00  24
0064: 91 0D 20 11  C9  24 00 00

006c: ######################## SP_DCSQT:
006c: ######################## SP_DCSQ 0 = 0x0
006c:  00 00                   # SP_DCSQ_STM = 0       = 0x0 = 0.0000s  => start frame:  0
006E:  00 64                   # SP_NXT_DCSQ_SA = 100 = 0x64
0070:  03  32  10              # SET_COLOR b=0 p=1 e1=2 e2=3
0073:  04  FF  FF              # SET_CONTR  (0..15) b=15 p=15 e1=15 e2=15
0076: 05 15 21 7D 19 01 9C     # SET_DAREA sx=338 ex=381 sy=400 ey=412
007D:  06 00 04 00 26          # SET_DSPXA uf=0x4     bf=0x26
0082:  01                      # STA_DSP
0083:  FF                      # CMD_END
0084: ######################## SP_DCSQ 1 = 0x1
0084:  01  B0                  # SP_DCSQ_STM = 0x1B0 = 122.8800s
0084:                          #            => start frame:  123  ( PAL )
0086:  00 64                   # SP_NXT_DCSQ_SA = 100 = 0x64
0088:  02                      # STP_DSP
0089:  FF                      # CMD_END

/ ******************  THE RESULTING DECOMPRESSED SP:  **************************/

The following simple graphic shows the presented (decoded) bitmap.  The symbol
'.' indicates a background pixel and the symbol '#' indicates a pattern
pixel.  The following bitmap represents a sub-picture with horizontal 44 pixel
and vertical 13 pixel.

The  (simple)  graphic:

....................................................
....................................................
....................................................
.............##.....#.........##.....##.............
...........#...#...##.......#...#...#...#...........
...........#...#.#.#...#.....#.....#.....#...........
...........#...#.....#.....#.....#.....#.....#.....
...........#...#.....#...#.....#.....#.....#.....#.....
...........#...#.....#.....#.....#...#.....#.....
.............##.....###.....####....##.....
....................................................
....................................................
....................................................
....................................................
....................................................
....................................................
```

Fig.12

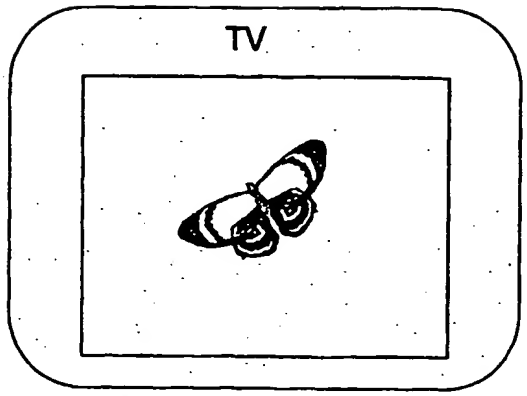

Fig.13a

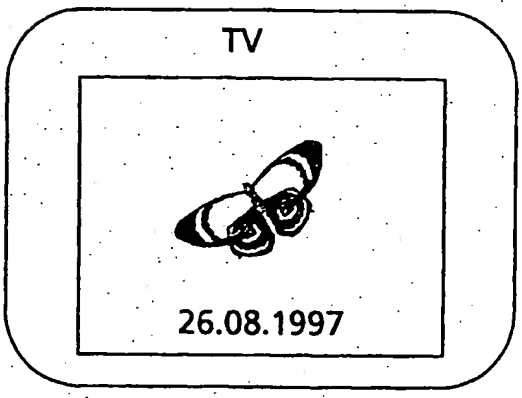

Fig.13b

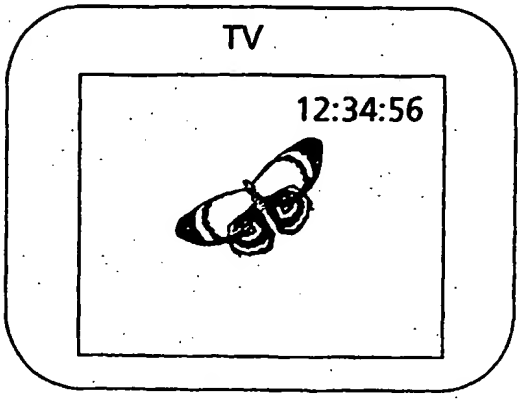

Fig.13c

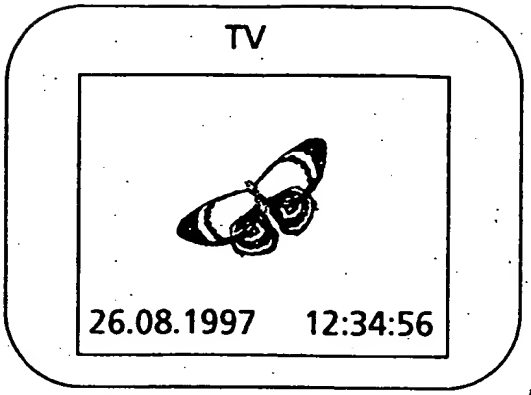

Fig.13d

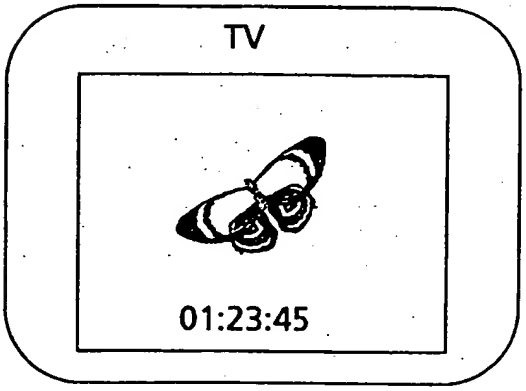

Fig.13e

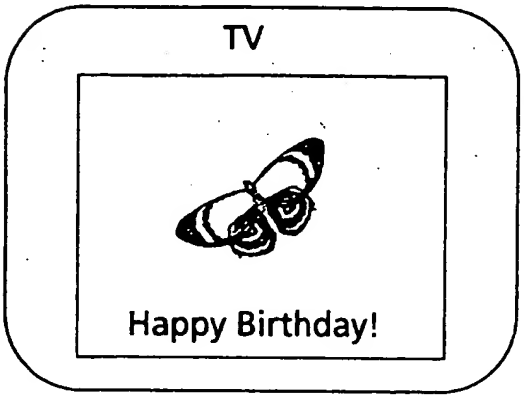

Fig.13f

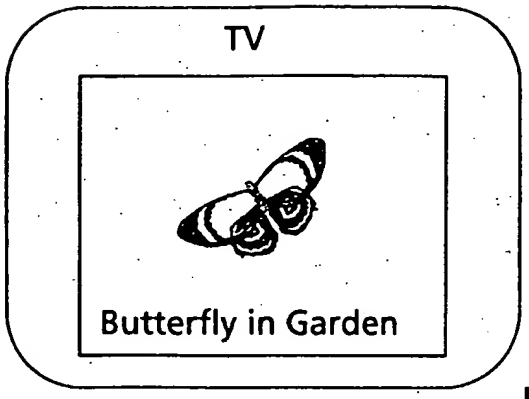

Fig.13g

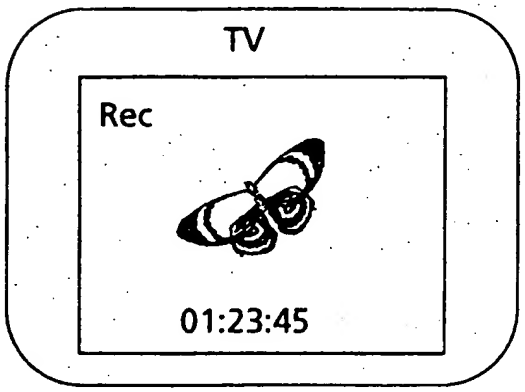

Fig.13h

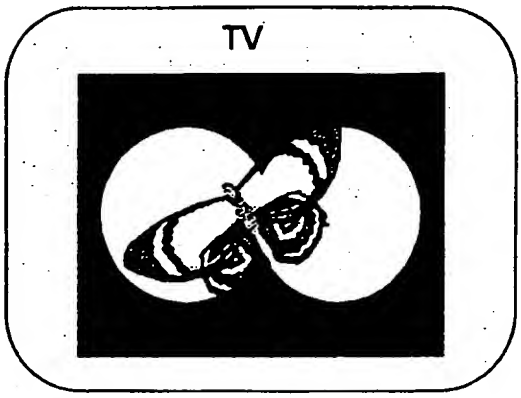

Fig.13i

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.